

EXPRESS MAIL LABEL NO.: ET944327195US DATE OF DEPOSIT: 12/21/2001

I hereby certify that this paper and fee are being deposited with the United States Postal Service Express Mail Post Office to Addressee service under 37 CFR § 1.10 on the date indicated below and is addressed to the Commissioner of Patents, Washington, D.C. 20231

CATHERINE M. ROBBINS

NAME OF PERSON MAILING PAPER AND FEE

Catherine M. Robbins
SIGNATURE OF PERSON MAILING PAPER AND FEE

Inventor(s): Timo J. Salo
Kevin J. Williams

DECENTRALIZED MANY-TO-MANY RELATIONSHIP MANAGEMENT IN AN OBJECT PERSISTENCE MANAGEMENT SYSTEM

BACKGROUND OF THE INVENTION

Statement of the Technical Field

The present invention relates to the field of object state persistence and more particularly to many-to-many relationship management in an object persistence management system.

Description of the Related Art

As object-oriented technology has matured, it has proven to be an excellent solution for modeling problems, building prototypes, and rapidly deploying applications. Though object models for applications are often reused in other applications, one of the costlier tasks of development has been that of translating between object-oriented and non-object-oriented representations of application models. Consequently, the task of mapping objects both to non object oriented relational database tables and other non object oriented data sources has been the missing piece in object persistence management.

Conventional object persistence management systems typically include a collection of development browsers, tools, and code generation services that assist in the development of persistence support within object-oriented applications. Browsers are the main resources for building applications using the persistence management framework. In particular, browsers are used to describe object models, database schema, and data store mappings, as well as to generate source code need by the framework to manage objects in the model. Conventional object persistence management systems can include several browsers, including object model browsers, schema browsers and map browsers.

Object browsers can be used to define object models, its classes and associations. Schema browsers, by comparison, can be used to define a logical description of the data store to which the object model will persist. Importantly, map browsers can be used to map the object model, or persistent classes, to the logical, database schema. In particular, each persistent class requires a map which associates the attributes of the class with their corresponding columns or fields in the database tables or records thereof.

In that regard, present object persistence management systems provide for one-to-one, one-to-many and many-to-many relationship management. In the case of one-to-one and one-to-many relationship management, present object persistence management systems provide decentralized, loosely coupled agents, referred to herein as "links", to manage each side of the relationship. In particular, each link manages the state of an object in the relationship and coordinates with a corresponding "counter-link" when necessary, as is well-known in the art.

Part of the relationship management process can include the management of the persistent state of the relationship. The management of the persistent state of the relationship can be relatively straightforward in the case of one-to-one and one-to-many relationships since the state of the relationship can be included as part of the state of the related objects themselves. Hence, if the links have correctly managed the related objects, when the objects are stored to an underlying database, the state of the relationship also will be stored.

A typical method used to store or persist the state of objects and their relationships in a one-to-one or one-to-many relationship can include using tables in a relationship database--one for each object in the relationship. In each table, a particular field can be used as a state reference in the other tables. For example, consider an object model of a class and an instructor having a one-to-one relationship with one another. A table can be constructed for all possible instructors, each instructor having a unique ID. A corresponding table can be constructed for all classes, each class having a reference to the unique ID of its instructor.

In this example, the values for the class reference to a unique instructor ID can dynamically change according to the state of the associated class and instructor objects at the time those objects are stored in the database. If the relationship management system correctly maintains the state of the related objects as their relationships change, then the relationships can be correctly stored to the underlying database when the objects are saved to a data store. If the saved objects are later restored from the data store, then the relationships also can be correctly restored.

Unlike one-to-one and one-to-many relationships, storing the state of many-to-many relationships is not as straightforward. This is so because a third, auxiliary table, referred to hereinafter as a "junction table", typically is required to store the relationship information. For example, if multiple students are enrolled in multiple classes in an instructor-class-student model, a many-to-many relationship will have been created as between the students and their respective classes in which they are enrolled.

In consequence, a junction table unrelated to any particular object will be required in order to track those students enrolled in the multiplicity of classes at any one time. Specifically, each record in the junction table can include two fields linking the unique ID of a student with the unique ID of a class in which the student is enrolled. These records typically are referred to as "key-pair entries". Of course, the junction table can include multiple key-pair entries for each student, each key-pair entry reflecting only one of the several classes in which the student may be enrolled.

Since the management of many-to-many relationships requires the management of a third, auxiliary entity apart from the related entities, the links managing the relationship between the related objects must be concerned not only with the state of the related objects, but also with the state of the third, auxiliary entity. One common approach to solving the problem of managing the junction table includes introducing a relationship manager responsible for updating key-pair entries in the junction table. As links to either side of the relationship detect relationship changes, the links inform the relationship manager of the change. In consequence, the relationship manager can ensure that the junction table has been updated accordingly.

Significantly, the common solution to managing many-to-many relationships in an object model can require the addition of substantial infrastructure to the run-time portion of the object persistence management system. This is so because typical object persistence management systems are designed such that only application entities such as the links provide state data to be flushed to the database. Conversely, only application entities can be retrieved from the database. The relationship manager, however, is not an application entity. Rather, the relationship manager merely is an artifact required to record the relationship state for many-to-many associations.

The decentralization of the management of the junction table, heretofore, has been complicated by potential conflicts which can arise when persisting state changes in related objects to their associated object tables. For instance, links typically can buffer state changes prior to their persistence in a table. When buffering the addition of a relationship, the link can buffer the operation, "add [key-pair entry]". Conversely, when buffering the removal of a relationship, the link can buffer the operation, "remove [key-pair entry]".

In both cases, however, it is possible that a corresponding link which manages the other portion of the relationship also can buffer the management operations pertaining to the key-pair entry. Specifically, it is possible that one link which manages the state of object1 buffers the operation, "add [object1, object2]", while the link which manages the state of object 2 buffers the operation, "remove[object1, object2]". When persisting the state of object1 and object 2, however, the timing of performing each buffered operation can undermine the integrity of the junction table.

Specifically, where the buffered operations of the link which manages object 2 are performed prior to the buffered operations of the link which manages object1, a conflict can arise. In consequence, despite the apparent disadvantages of the centralized management of the junction table, the relationship manager can be required to avoid such conflicts. Thus, there exists a need for a many-to-many relationship management system in which timing conflicts can be managed notwithstanding the absence of a centralized relationship manager.

5

10026387 123101

SUMMARY OF THE INVENTION

The present invention is a many-to-many relationship management system which overcomes the deficiencies of the prior art many-to-many relationship management systems which depend upon an auxiliary relationship manager. In the object persistence management system of the present invention, a many-to-many relationship manager can include one or more related objects; a junction table storing relationships between the related objects; and, one or more corresponding links.

Each link can correspond to one of the objects. Furthermore, each link can persist state information for the corresponding object in an associated object table. Finally, each link can manage the junction table responsive to changing relationships with others of the related objects. Importantly, as the present invention distributes the management of the junction table, a counter-operation management protocol can be provided which can resolve conflicts which arise in the management of the junction table in response to changing relationships among their associated objects.

The counter-operation management protocol can remove conflicted state information in the corresponding links without persisting the conflicted state information in the junction table. In particular, each of the corresponding links can include a state management operations buffer, the buffer storing directives for adding selected key-pair entries to and removing selected key-pair entries from the junction table. The counter-operation management protocol can include an interface through which operations in the buffer and corresponding counter-operations in associated buffers of related links can be identified and removed. Specifically, each counter-operation can specify a junction table management operation for a particular key-pair entry in the

associated buffer which is opposite to an operation in the buffer which specifies a junction table management operation also for the particular key-pair entry.

In one aspect of the invention, a method of managing a many-to-many relationship in an object persistence management system can include detecting a relationship change with a related object. Upon detecting a relationship change, a directive can be stored in a buffer, the directive specifying a management operation for changing the relationship in a junction table. Notably, the stored directive can be performed only if an opposite directive has not been stored in a buffer associated with the related object.

Notably, the storing step can include storing a directive in the buffer which specifies one of adding or removing a key-pair entry in the junction table. In that regard, the performing step can include performing the specified adding or removing of the key-pair entry only if a corresponding opposite directive specifying a respective removing or adding of the key-pair entry is not detected in the buffer of the associated object. In that regard, if a corresponding opposite directive is detected, both the directive and opposite directive can be removed from both of the buffers.

REPORT

5

WP070177;1

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

10026307-122101
TOTAL 4892001

The present invention is a decentralized many-to-many relationship management system. In the many-to-many relationship management system of the present invention, links which traditionally manage the objects in a one-to-one and one-to-many relationship, also can manage the related objects in a many-to-many relationship. Specifically, the links can collaborate with one another by providing updates to the junction table without the assistance of a non-application, auxiliary relationship manager. Notably, unlike conventional many-to-many relationship management systems in which a centralized relationship manager manages the junction table, in the present invention, the responsibility for maintaining the junction table can be distributed amongst the links.

Each of the links in the many-to-many relationship management system of the present invention can have an association with an object as the persistence manager for that object and, as such, can be considered an application entity. In consequence, the persistent state information managed by each link can be stored to and retrieved from data storage using the conventional flush and hydration mechanisms of pre-existing conventional object persistence management systems. Thus, by eliminating the need for a centralized relationship manager, the complexity of managing many-to-many relationships can be dramatically reduced.

Finally, inasmuch as a conflict of flush and hydration operations can arise on opposite sides of a many-to-many association, a counter-operation process can be provided in accordance with the present invention. The counter-operation process can be performed in each of the links in the many-to-many relationship prior to the

persistence of object states managed by each side of the many-to-many association.

Specifically, prior to performing individual flush and hydration operations wherein the

state of the managed objects are written to or deleted from their associated object

tables, each managing link can inspect its own operational buffer and the operational

5 buffer of the another to locate operations in both which run counter to one another.

Where identified, these operations can be removed from the buffer without performing

the corresponding individual flush or hydration operation. In this way, potential conflicts

within the many-to-many relationship can be avoided.

Figure 1 is a schematic illustration of the decentralized many-to-many

10 relationship management system of the present invention. As illustrated in Figure 1,

the system can include two or more objects 108, 110 having a many-to-many object

relationship. An example of two such objects can include the object relationship

between classes in which students can enroll, and students who can enroll in classes.

Other examples can include the object relationship between projects to which

15 employees can be assigned, and employees who can work on projects.

In accordance with the inventive arrangements, an object persistence

management system 100 can be included which can ensure the persistence of the

object state in each object 108, 110 managed by the system 100. More particularly,

links 104, 106 can be provided which can ensure the integrity of the state of the related

20 objects 108, 110 in the many-to-many relationship. The links 104, 106 can be

application entities which coordinate state changes among the related objects 108, 110

and whose coordinated states can be flushed to a data store and hydrated from a data

store, for instance tables 112, 114.

The many-to-many relationships can be recorded in a junction table 102. Specifically, the junction table 102 can store the state of the many-to-many relationship between the objects 108, 110 as is the case in a conventional junction table. Importantly, however, a centralized relationship manager does not manage the addition and deletion of data to and from the junction table 102, respectively. Rather, in the present invention each link 104, 106 of the many-to-many relationship management system can manage the auxiliary table 102, thus eliminating the requirement that an artifact, non-application entity manage the auxiliary table 102. Specifically, the links 104, 106 can manage the auxiliary table 102 according to an agreed-upon management protocol.

Figure 2 is a flow chart illustrating the management of state changes in a primary link the decentralized many-to-many relationship management system of Figure 1. Importantly, by "primary link", it is meant that the object managed by the primary link enjoys a many-to-many relationship with other objects managed by corresponding secondary links. Of course, as one skilled in the art will recognize, each secondary link, from the perspective of their managed objects, in of themselves can be primary links. Hence, the term primary link is referred to herein only as a matter of semantic convenience and not as a matter of operational distinction between the objects in a many-to-many relationship.

Beginning in block 202, the primary link can detect a request to change the state of its associated object by altering the object's relationship with another related object. For instance, the request can be a request to add a new relationship as in the case of enrolling a student in a new class (Class.Add(Student) or Student.Add(Class)), or

removing an employee from an existing project (Project.Remove(Employee) or Employee.Remove(Project)). In any case, in decision block 204, it can be determined whether the request is one to add or delete a relationship.

If the request is a request to add a new relationship, in block 206 the primary link
5 can change its internal state to reflect the new relationship with the related object. Specifically, the link can buffer the operation, "add [new key-pair entry]" to reflect the new relationship. By comparison, if in decision block 204 the request to change the object state is determined to be a request to delete a relationship between two objects,
10 in block 208 the primary link can change its internal state to remove the relationship with the related object managed by the secondary link. In particular, as before the primary link can buffer the operation, "remove [existing key-pair entry]".

In decision block 210, if the state of object managed by the primary link is not to be persisted, the process can continue in block 202 wherein additional object state changes can be processed in the primary link. If, however, in decision block 210, the
15 state of the object managed by the primary link is to be persisted, in block 212, the buffers of each secondary link associated with the primary link in a many-to-many relationship can be inspected for counter-operations to those operations stored in the buffer of the primary link.

Where counter-operations in a secondary link are identified, both the primary link
20 and secondary link can remove the operation and corresponding counter-operation from their respective buffers in step 214. Finally, in step 216, those remaining operations in the buffer of the primary link can be persisted to the junction table. Notably, though the counter-operation process of Figure 2 is shown to occur

subsequent to a decision to persist the state of the link, in a preferred embodiment, the counter-operation process illustrated in steps 212 through 216 can be performed in response to a decision to add or remove an operation to the link buffer.

Importantly, because the management of the junction table has been decentralized in the present invention, the need for a centralized relationship manager has been eliminated in accordance with the inventive arrangements. Yet, the conflicts which can arise from decentralized management of the junction table can be avoided in the present invention by virtue of the counter-operation process performed prior to the persistence of object states in the junction table. As a result, the complexity of the object persistence management system can be significantly reduced.

The present invention can be realized in hardware, software, or a combination of hardware and software. An implementation of the method and system of the present invention can be realized in a centralized fashion in one computer system, or in a distributed fashion where different elements are spread across several interconnected computer systems. Any kind of computer system, or other apparatus adapted for carrying out the methods described herein, is suited to perform the functions described herein.

A typical combination of hardware and software could be a general purpose computer system with a computer program that, when being loaded and executed, controls the computer system such that it carries out the methods described herein. The present invention can also be embedded in a computer program product, which comprises all the features enabling the implementation of the methods described

herein, and which, when loaded in a computer system is able to carry out these methods.

Computer program or application in the present context means any expression, in any language, code or notation, of a set of instructions intended to cause a system
5 having an information processing capability to perform a particular function either directly or after either or both of the following a) conversion to another language, code or notation; b) reproduction in a different material form. Significantly, this invention can be embodied in other specific forms without departing from the spirit or essential attributes thereof, and accordingly, reference should be had to the following claims,
10 rather than to the foregoing specification, as indicating the scope of the invention.

10026387-132101
TOTAT" 2809200T